Part 03
-
WiFi

Version: 2021-06-01

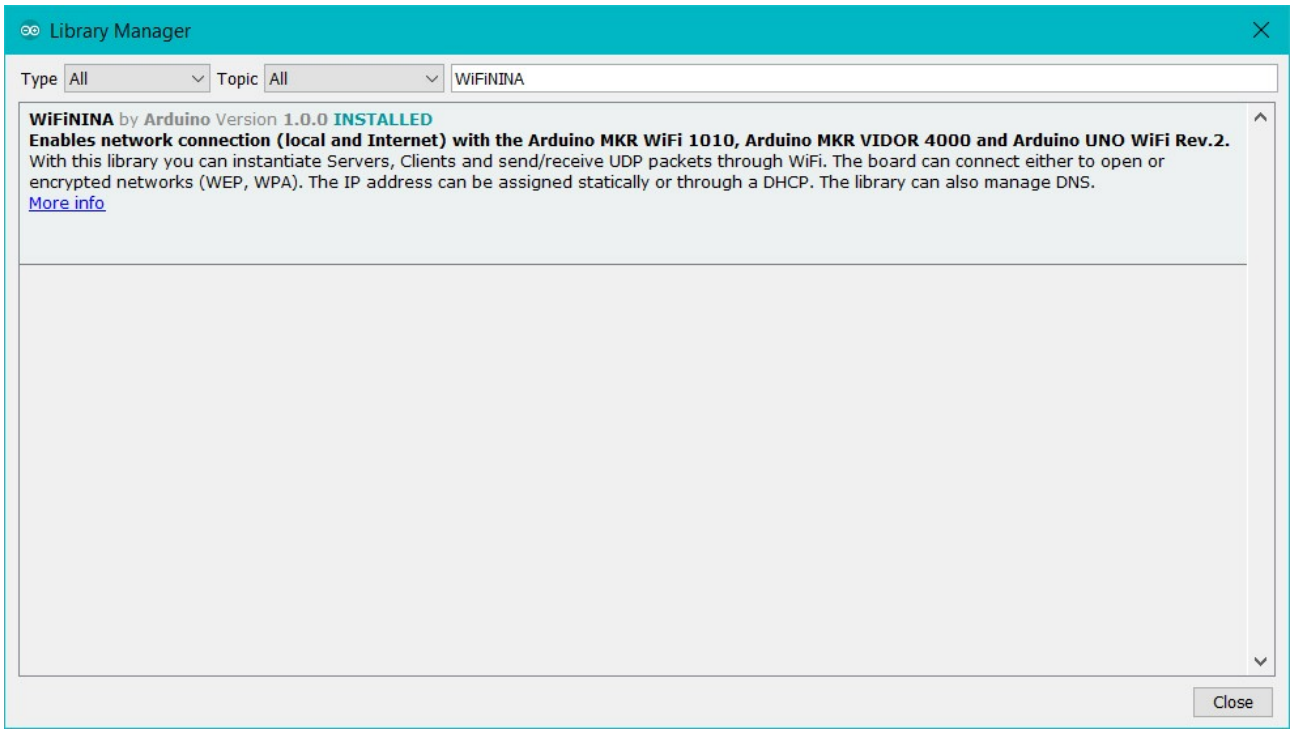
WiFiNINA library

This library allows you to use the WiFi capabilities. It can serve as either a server accepting incoming connections or a client making outgoing ones. The library supports WEP, WPA2 Personal and WPA2 Enterprise encryptions. This library support all the same methods of the original WiFi library plus the `connectSSL()`. The WiFiNINA library is very similar to the Ethernet and the library WiFi, and many of the function calls are the same.

To use this library

```
#include <SPI.h>
#include <WiFiNINA.h>
```

This library requires that your board has a matching firmware installed. When the library is updated, also the firmware might be updated, but it is not mandatory. To avoid any issue and ensure that you have the most up to date setup, we suggest that you check your WiFiNINA library with the Arduino Software (IDE) Library Manager. There is an option in the Preferences that enables the check for updates of any of the installed libraries at startup. If you haven't installed the WiFiNINA library yet, you won't get notified about its updates. Anyway, you get the library status just writing its name in the search field on top of the Library Manager.



When the library version installed on your computer is the latest available, you may check the firmware version of the board or the shield. We have prepared a utility sketch to check the firmware version and its matching with the library. If the firmware needs an update, another utility sketch enables the process. You find these two utilities under `Examples -> WiFiNINA -> Tools`.

- CheckWiFiNINAFirmwareVersion : Reads the required firmware number required from library and matches with the one installed on the board or the shield.
- WiFiNINAFirmwareUpdater : The sketch that must be loaded to allow the firmware and certificates update process through the integrated plugin of Arduino Software (IDE) rel. 1.8.5 or later.

Web Server
The Nano RP2040 Connect features a Wi-Fi module and an RGB LED among many other things. In this tutorial we will take a look at how we turn our board into an access point, and control the built-in RGB through a browser connected to the access point.

Controlling over Wi-Fi
There are multiple ways we can access our board over Wi-Fi. In this tutorial, we will turn our board into a web server, that will listen for incoming GET requests.
Simply explained, we will set up our board as an access point, and start hosting a web server on a specific IP address. The device will now show up in the list of available Wi-Fi networks on for example your smarthpone or computer.
If we connect to this access point, and enter the board's address in the browser of a computer/browser on the same network, we make a request to this server. The server responds with a set of HTML instructions, which can then be viewed in the browser.

Controlling through the browser
Inside the HTML instructions that we print, or send to the browser, we can create **buttons.** These buttons can be modified to add something to the URL. Here is an example of how a button is created:

```
<button class='red' type='submit' onmousedown='location.href=\"/RH\"'>ON</
button>
```

The most important is what we put inside href, which in this case is /RH. Whenever this button is clicked, it will update the URL of the page to http://<board-ip>/RH. After this happens, the program will go through a set of conditionals, that checks whether this button has been pressed:

```
if (currentLine.endsWith("GET /RH"))
{
   doSomething();
}
```

By using this method, we can set up many more buttons that can control different aspects of our Arduino, and is a great building block for creating a control interface for your board that can be controlled through a browser, over Wi-Fi.

Programming the board
We will now get to the programming part of this tutorial.

1. First, let's make sure we have the drivers installed. If we are using the Web Editor, we do not need to install anything. If we are using an offline editor, we need to install it manually. This can be done by navigating to **Tools > Board > Board Manager...**. Here we need to look for the **Arduino Mbed OS Nano Boards** and install it.

2. Now, we need to install the libraries needed. If we are using the Web Editor, there is no need to install anything. If we are using an offline editor, simply go to **Tools > Manage libraries..**, and search for **WiFiNINA** and install it.

3. We can now take a look at some of the core functions of this sketch:
   - create a network name and network password
     ```
     char ssid[] = ""
     char pass[] = ""
     ```
   - creates an access point with the stored credentials.
     ```
     WiFi.beginAP(ssid, pass)
     ```
   - creates a server that listens for incoming connections on the specified port.
     ```
     WiFiServer server(80)
     ```
   - creates a client that can connect to to a specified internet IP address.

```
WiFiClient client
```

- tells the server to begin listening for incoming connections.
- `server.begin()`

- checks for connected clients.
  `client.connected`

- checks for available data.
  `client.available`

- reads the available data.
  `client.read`

- print something to the client (e.g. html code).
  `client.print()`

- closes the connection.
  `client.stop()`

The sketch can be found in the snippet below. Upload the sketch to the board.
Note: both ssid and pass needs to be a minimum of 8 characters long.

```
#include <SPI.h>
#include <WiFiNINA.h>

char ssid[] = "yournetwork";   // network SSID
char pass[] = "yourpassword";  // network password
int keyIndex = 0;              // network key index number needed for WEP

int status = WL_IDLE_STATUS;
WiFiServer server(80);

void setup()
{
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial)
  {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.println("Access Point Web Server");

  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);

  // check for the WiFi module:
  if (WiFi.status() == WL_NO_MODULE)
  {
    Serial.println("Communication with WiFi module failed!");
    // don't continue
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION)
  {
    Serial.println("Please upgrade the firmware");
  }

  // by default the local IP address will be 192.168.4.1
  // you can override it with the following:
  // WiFi.config(IPAddress(10, 0, 0, 1));

  // print the network name (SSID);
  Serial.print("Creating access point named: ");
  Serial.println(ssid);

  // Create open network.
  // Change this line if you want to create an WEP network:
  status = WiFi.beginAP(ssid, pass);
  if (status != WL_AP_LISTENING)
  {
    Serial.println("Creating access point failed");
    // don't continue
    while (true);
  }

  // wait 10 seconds for connection:
  delay(10000);
```

```
  // start the web server on port 80
  server.begin();

  // you're connected now, so print out the status
  printWiFiStatus();
}


void loop()
{
  // compare the previous status to the current status
  if (status != WiFi.status())
  {
    // it has changed update the variable
    status = WiFi.status();

    if (status == WL_AP_CONNECTED)
    {
      // a device has connected to the AP
      Serial.println("Device connected to AP");
    }
    else
    {
      // a device has disconnected from the AP
      // and we are back in listening mode
      Serial.println("Device disconnected from AP");
    }
  }

  WiFiClient client = server.available(); // listen for incoming clients

  // if you get a client,
  if (client)
  {
    Serial.println("new client");   // print a message out the serial port
    String currentLine = "";        // make a String to hold incoming data
    // loop while the client's connected
    while (client.connected())
    {
      // if there's bytes to read from the client,
      if (client.available())
      {
        char c = client.read();     // read a byte, then
        Serial.write(c);            // print it out the serial monitor
         // if the byte is a newline character
        if (c == '\n')
        {

          // if the current line is blank
          // you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0)
          {
            // HTTP headers always start with a response code
            // (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming
            // then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // the content of the HTTP response follows the header:
            client.print("<style>");
            client.print(".container {margin: 0 auto; text-align: center;
```

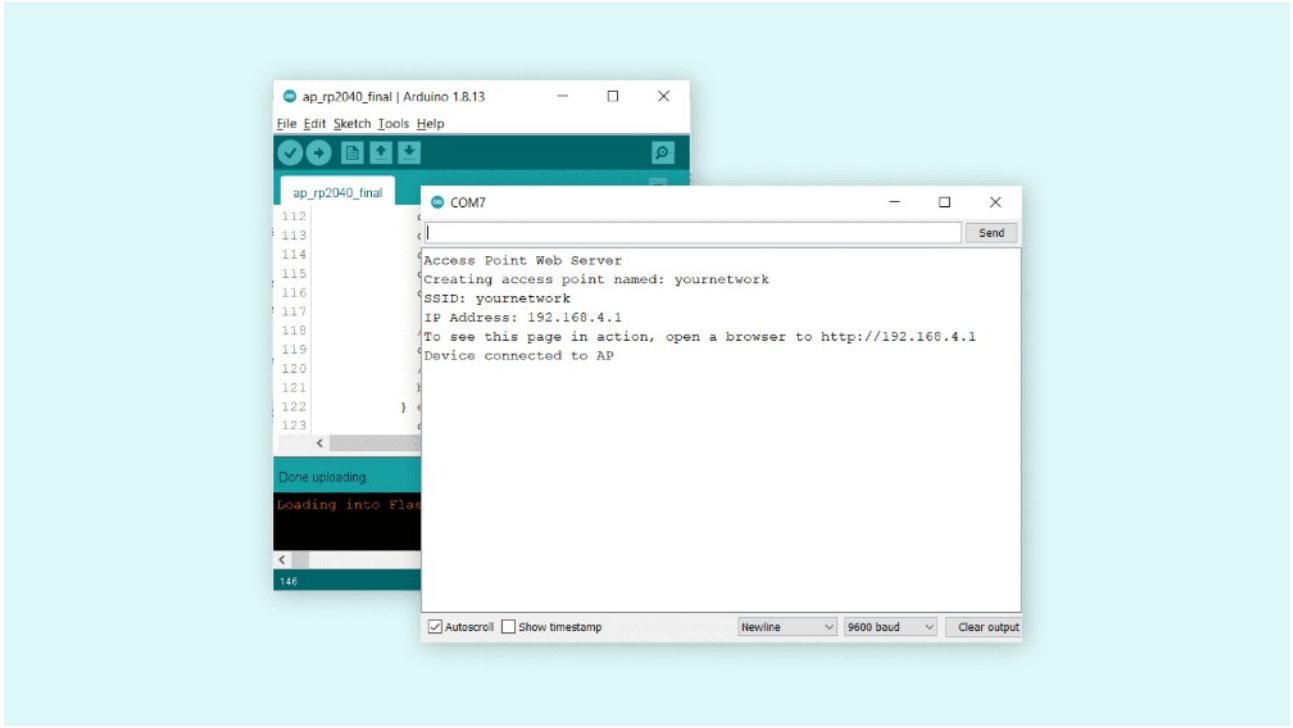
```
margin-top: 100px;}");
            client.print("button {color: white; width: 100px; height:
100px;");
            client.print("border-radius: 50%; margin: 20px; border: none;
font-size: 20px; outline: none; transition: all 0.2s;}");
            client.print(".red{background-color: rgb(196, 39, 39);}");
            client.print(".green{background-color: rgb(39, 121, 39);}");
            client.print(".blue {background-color: rgb(5, 87, 180);}");
            client.print(".off{background-color: grey;}");
            client.print("button:hover{cursor: pointer; opacity: 0.7;}");
            client.print("</style>");
            client.print("<div class='container'>");
            client.print("<button class='red' type='submit' onmousedown='loc-
ation.href=\"/RH\"'>ON</button>");
            client.print("<button class='off' type='submit' onmousedown='loc-
ation.href=\"/RL\"'>OFF</button><br>");
            client.print("<button class='green' type='submit'
onmousedown='location.href=\"/GH\"'>ON</button>");
            client.print("<button class='off' type='submit' onmousedown='loc-
ation.href=\"/GL\"'>OFF</button><br>");
            client.print("<button class='blue' type='submit'
onmousedown='location.href=\"/BH\"'>ON</button>");
            client.print("<button class='off' type='submit' onmousedown='loc-
ation.href=\"/BL\"'>OFF</button>");
            client.print("</div>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          }
          else
          {    // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        }
        else if (c != '\r')
        {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }

        // Check to see if the client request was /X
        if (currentLine.endsWith("GET /RH"))
        {
          digitalWrite(LEDR, HIGH);
        }
        if (currentLine.endsWith("GET /RL"))
        {
          digitalWrite(LEDR, LOW);
        }
        if (currentLine.endsWith("GET /GH"))
        {
          digitalWrite(LEDG, HIGH);
        }
        if (currentLine.endsWith("GET /GL"))
        {
          digitalWrite(LEDG, LOW);
        }
        if (currentLine.endsWith("GET /BH"))
        {
          digitalWrite(LEDB, HIGH);
        }
        if (currentLine.endsWith("GET /BL"))
        {
```

```
          digitalWrite(LEDB, LOW);
        }
      }
    }
    // close the connection:
    client.stop();
    Serial.println("client disconnected");
  }
}

void printWiFiStatus()
{
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print where to go in a browser:
  Serial.print("To see this page in action, open a browser to http://");
  Serial.println(ip);

}
```

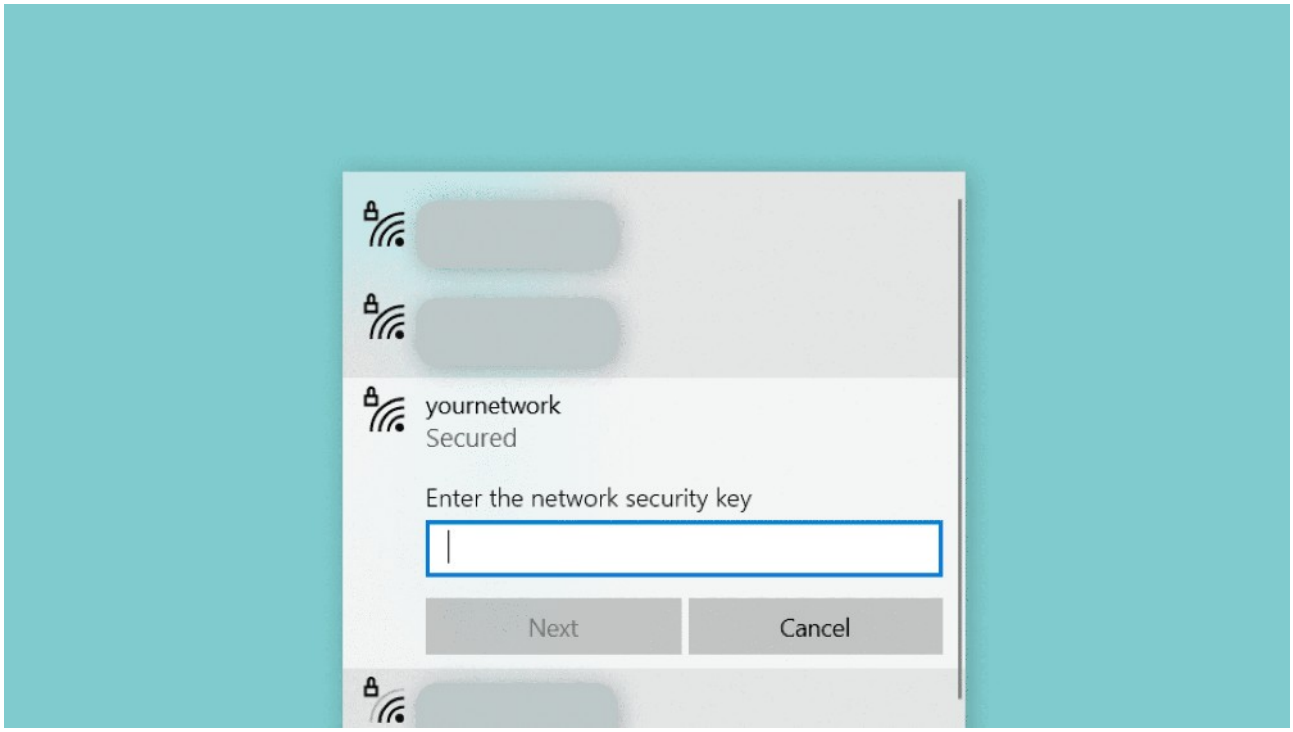
Testing it out

After the code has been successfully uploaded to the board, we need to open the Serial Monitor to initialize the program.

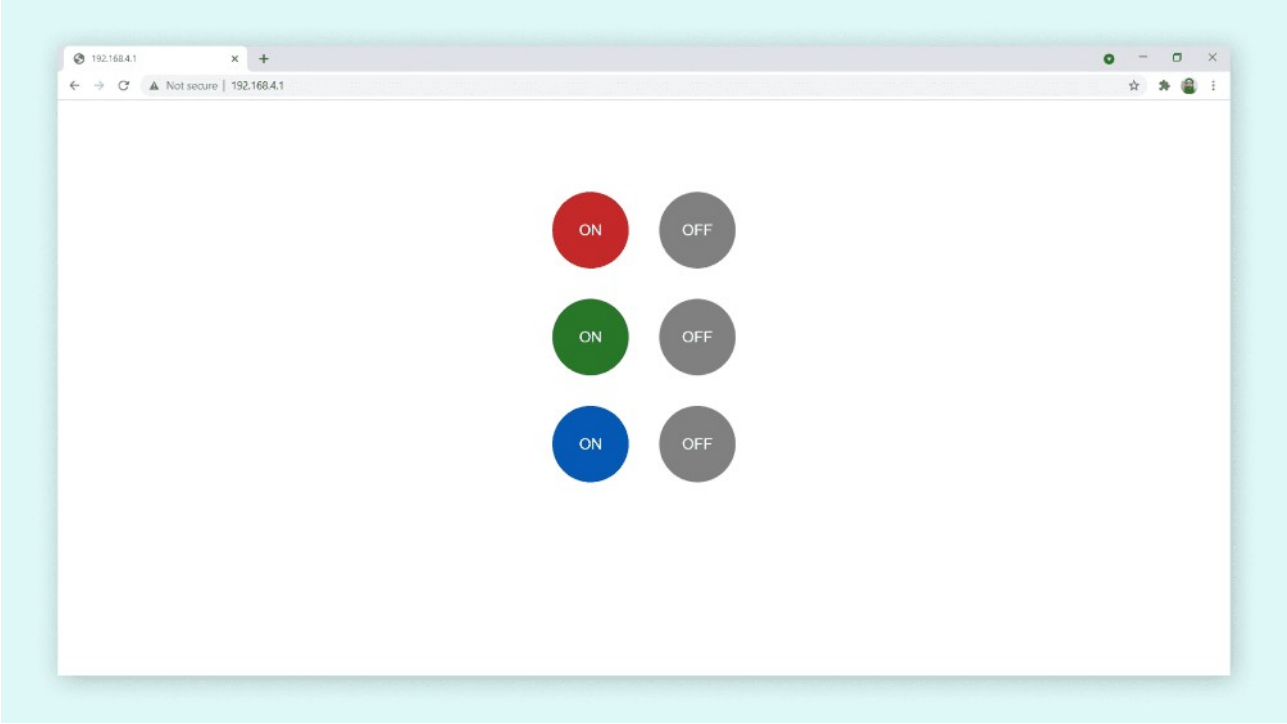
When we open it, it will attempt to create an access point, and if it is successful, it will print out the boards IP address in the Serial Monitor.



We will now need to connect to this access point through either our smartphone or computer, by browsing the list of available networks.



Once we have connected, we need to copy the enter the address that was printed in the Serial Monitor, in the browser of our device. We should now see a web page with a set of colored buttons.



We can now interact with the different buttons. The buttons available are used to control the built-in RGB LED on the Nano RP2040 Connect. There's six buttons in total, three to turn ON the different colors, and three to turn them off.

If the code is not working, there are some common issues we can troubleshoot:

- We have entered the wrong credentials to our Wi-Fi network.
- We have not installed the WiFiNINA library.
- The WiFiNINA library is updated (can be done in the library manager).